Note - Agar left to right jate hue elements ka track rkhna ho, then vha pr queue use kr skte hai. __/__/____

In below question, isi pattern ka use kiya hai.

* First non-repeating character in a stream -

I/P - "abacbdcd"

O/P - "aabbccd#"

	a	b	a	c	b	d	c	d
i →	0	1	2	3	4	5	6	7

First non-repeating characters:

i = 0 → "a" → ans = 'a'
i = 1 → "ab" → ans = 'a'
i = 2 → "aba" → ans = 'b'
i = 3 → "abac" → ans = 'b'
i = 4 → "abacb" → ans = 'c'
i = 5 → "abacbd" → ans = 'c'
i = 6 → "abacbdc" → ans = 'd'
i = 7 → "abacbdcd" → ans = '#'

If there is no first non-repeating character, then ans is '#'.

→ So, here, at every instance, we should know the frequency of each characters in stream.

__/__/____

Approach-

- Make a frequency table using array or map. Ye frequency table track rkhega ki konsa character kitni baar occur hua hai.
- Make a queue jo current stream ke liye kya answer hoga ye btayegi.

Step 1- String mei se character nikalo.

Step 2- frequency table mei us character ki occurrence ko increment kro.

Step 3- Us character ko queue mei push krdo.

Step 4- Jb tak queue empty nhi ho jati, tb tak check kro.
if front character hai queue mei uski frequency 1 hai. If yes, use o/p string mei push krvado. else queue mei se pop krdo us character ko.

Step 5- Agr koi answer nhi nikla or queue empty ho gyi, then o/p string mei '#' push krdo.

let the input → bcbacdae

queue = [b|c|b|a|c|d|a|e]

frequency table
b = 2
c = 2
a = 2
d = 1

o/p = bbccaadd#

Code-

```
class Solution{
    public:
    string FirstNonRepeating(string A){
        string s;
        int freq[26]={0};
        queue<char> q;

        for(int i=0; i<A.length(); i++){
            freq[A[i]-'a']++;
            q.push(A[i]);

            while(!q.empty()){
                if(freq[q.front()-'a']==1){
                    s.push_back(q.front());
                    break;
                }
                else{
                    q.pop();
                }
            }
            if(q.empty()){
                s.push_back('#');
            }
        }
        return s;
    }
};
```

__/__/___

* Gas Station (leetcode - 134)

There are n gas stations along a circular route where amount of gas at ith station is gas[i]. You have a car with unlimited gas tank and it costs cost[i] of gas to travel from ith station to its next (i+1)th station. You begin journey with an empty tank at one of the gas stations. Given two arrays gas and cost, return the starting station's index if you can travel around the circuit once in clockwise direction, otherwise return -1. If there exists a solution, it is guaranteed to be unique.

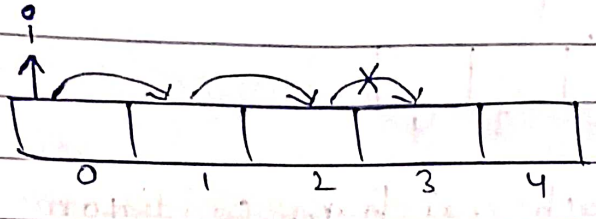
I/P - gas = [5, 1, 2, 3, 4] cost/distance = [4, 4, 1, 5, 1]

O/P - 4

Brute force approach -

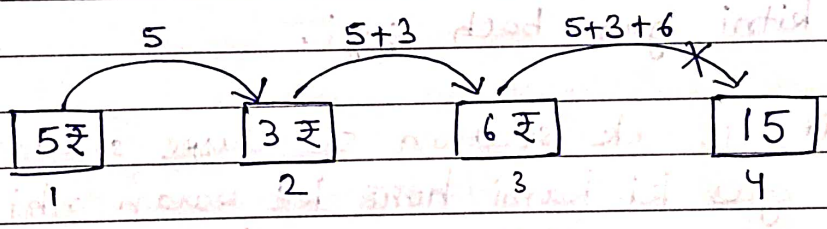
Har gas station pr jake check krlo ki next gas station pr ja payenge ya nhi.
Jis gas station se shuru kiya hai agr us station pr vapis pahuch paye, it means vhi index answer hai.

T.C → $O(n^2)$ which is not good.

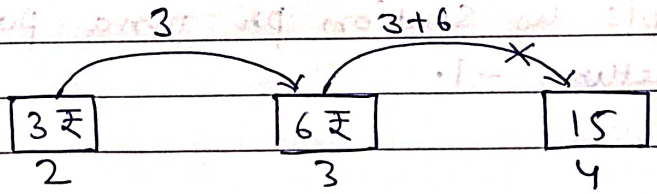


Agr 0 se 1st index pr gye, 1st se IInd index pr gye but 2nd se 3rd index pr nhi ja paaye.
Then, brute force approach mei hum ab 1st index se try krenge ki agr 1st index se shuru kiya, then vapis first index pr pahuche ki nhi but if agr hum 0th index ke contribution ke sath nhi pahuch paye, then uske bina contribution ke bhi nhi pahuch payenge.

let - we have 4 boxes. The 4th box needs 15 rs.

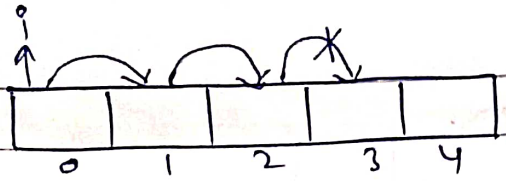


So, we can see ki ye 3 boxes milke 4th box ko 15 rs nhi de paye.
If mei 1st box ko remove krdo -



Is case mei bhi ye 15 rs ka amount nhi collect kr payenge.

Conclusion is, agr hum 1st box ke contribution ke sath 4th box ko reach nhi kr paye, then uske contribution ke bina bhi hum 4th box ko reach nhi kr payenge.



Optimized approach mei hum condition break hone par 0th index ke bad 1st index se nhi shuru krenge.. Vha se shuru krenge jha par condition break hui hai i.e. 3rd index se.

Imp point - Jha condition break hui hai vha se start krna hai.

In the code →

= surplus is ek station se dusre station pr jane mei kitni gas bach gyi.

= deficit is ek station se dusree station pr
- kitni gas ki kami hone ke karan nhi ja paye.
= pichla sara deficit calculate kr lenge.
= Total surplus calculate kr lenge.

Agr surplus >= deficit, it means us station se vapis us station pr jana possible hai.
else return -1.

Code-

```
class Solution {
public:
    int canCompleteCircuit (vector<int>& gas, vector<int>
                                                &cost)
    {
        int surplus= 0;
        int deficit = 0;
        int start = 0;

        for(int i=0; i< gas.size(); i++) {
            surplus += gas[i]- cost[i];
            if(surplus < 0) {
                deficit += abs(surplus);
                start = i+1;
                surplus = 0;
            }
        }
        if ( surplus - deficit >= 0) return start;
        return -1;
    }
};
```

* **Sliding Window Maximum (leetcode - 239)**

I/P - nums = [1, 3, -1, -3, 5, 3, 6, 7] , k = 3

O/P - [3, 3, 5, 5, 6, 7]

Explanation -

				Max
ω_1 =	1	3	-1	3
ω_2 =	3	-1	-3	3
ω_3 =	-1	-3	5	5
ω_4 =	-3	5	3	5
ω_5 =	5	3	6	6
ω_6 =	3	6	7	7

Approach -

- First window ko process krlo queue mei initial state maintain krne ke liye.
- Remaining windows ko process krlo.

Sliding window →
- Process first window
- Process remaining windows
  - store answer
  - Removal
  - Addition
- store answer of last window

Removal → (1) out of range elements
(2) Current element se chotte element in queue

Addition → Saare indices push honge in queue one by one.

→ Queue mei hum decreasing order ko maintain krke chl rahe hai.

Code -

```
class Solution {
public:
    vector<int>maxSlidingWindow(vector<int>&nums, int k)
    {
        vector<int>ans;
        deque<int> dq;
    //process first window
     for(int i=0; i<k; i++){
          int element = nums[i];
          //removal
          while(!dq.empty() && element>nums[dq.back()])
           {
                dq.pop_back();
           }
          //addition
          dq.push_back(i);

    //process remaining windows
     for(int i=k; i<nums.size(); i++){
         ans.push_back(nums[dq.front()]);

        //remove - out of range elements
        if(!dq.empty() && i-dq.front()==k){
              dq.pop_front();
            }
        //remove - chotte elements
          int element = nums[i];
          while(!dq.empty() && element>nums[dq.back()]){
                dq.pop_back();
              }
        //addition
          dq.push_back(i);
    }
```

```
        ans.push_back(nums[dq.front()]);
        return ans;
    }
};
```